Laboratory of Computer Vision

Lab: Practical work of computer vision for non-conventional cameras .

Author:

- **AMMAR KHODJA Rayane** 20225452@etud.univ-evry.fr
- **Daniel Shehroz khan SABIR JAMIL** 20235553@etud.univ-evry.fr

Supervised by:

- **Hicham HADJ-ABDELKADER**

Master MMVAI November 2024.
Paris Saclay University.

Laboratoire de Recherche Paris Saclay University, IBISC Evry Val d'Essonne.

Abstract

This report explores the integration of computer vision techniques in the context of non-conventional cameras, focusing on the conversion of 2D images to spherical coordinates. The study involves the detection of features in both 2D and spherical images, leveraging homography matrices to establish geometric relationships between the two domains.

The process begins with gray level scaling to enhance feature visibility in images, followed by feature detection using methods like Harris corner detection or SIFT. Subsequently, a homography matrix is computed to map corresponding points between 2D and spherical images, facilitating coordinate transformation.

The conversion of Cartesian coordinates to spherical coordinates allows for a seamless transition between different imaging perspectives. Feature matching is then performed in the spherical image space, enabling the identification of common features across both representations.

The report provides MATLAB code snippets illustrating the implementation of these steps, offering a practical guide for researchers and practitioners exploring non-conventional camera applications. The presented methodologies pave the way for comprehensive feature analysis and cross-domain image comparisons in the realm of computer vision and panoramic imaging.

1 Introduction

The advent of non-conventional cameras has opened new avenues in computer vision, challenging traditional imaging paradigms. This report delves into the transformation of 2D images into spherical coordinates, a pivotal task in panoramic imaging. The focus lies on feature detection, homography matrices, and coordinate conversion to bridge the gap between 2D and spherical representations.

A crucial aspect involves establishing geometric relationships between the 2D and spherical images through homography matrices. These matrices enable the seamless mapping of corresponding points, facilitating coordinate transformation. The transition from Cartesian to spherical coordinates serves as a crucial step in harmonizing the two imaging domains.

Theoretical Framework: Camera Projection and Calibration

Camera Projection:

In computer vision, camera projection is a fundamental concept that describes the mapping of 3D points in the world coordinates onto a 2D image plane. The projection process is mathematically represented by a projection matrix, often denoted as P. The projection matrix P is a 3x4 matrix that encapsulates both the intrinsic and extrinsic parameters of the camera.

Intrinsic Parameters:

1. **Focal Length (f):** The focal length represents the distance from the camera's optical center to the image plane. It is a critical intrinsic parameter affecting the field of view and image magnification.
2. **Principal Point (c_x, c_y):** The principal point denotes the coordinates of the optical center on the image plane. It is the point where the optical axis intersects the image plane.
3. **Pixel Aspect Ratio (α):** Accounts for potential distortion due to non-square pixels, ensuring accurate representation of objects in the image.

The intrinsic matrix K is expressed as:

$$K = \begin{bmatrix} f & 0 & c_x \\ 0 & \alpha f & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Camera Calibration:

Camera calibration is a crucial process that establishes the relationship between the 3D world coordinates and their corresponding 2D image projections. The calibration process aims to determine the intrinsic parameters of the camera, enabling accurate spatial measurements.

Single Viewpoint Constraint:

The Single Viewpoint Constraint asserts that all points in the 3D world are observed from a single viewpoint, simplifying the camera model. This constraint assumes a pinhole camera model, where light rays pass through a single point (the optical center) and project onto the image plane. The projection of a 3D point (X, Y, Z) to a 2D point (u, v) is governed by the projection matrix P:

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = P \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Here, w is a scaling factor ensuring homogeneous coordinates. The intrinsic matrix K and the extrinsic matrix $[R|t]$ together form the projection matrix P:

$$P = K[R|t]$$

Where:

- R is the rotation matrix representing the camera's orientation.
- t is the translation vector denoting the camera's position in 3D space.

Understanding the camera projection model and the intrinsic parameters is pivotal for accurate camera calibration, forming the cornerstone for subsequent transformations and analysis in computer vision applications.

Based research papers of our Project

1. **A Unifying Theory for Central Panoramic Systems and Practical Implications** Christopher Geyer and Kostas Daniilidis University of Pennsylvania, GRASP Laboratory, Pennsylvania, PA 19104 Research Paper in this link

2. **Points-Based Visual Servoing with Central Cameras** Hicham Hadj-Abdelkader, Youcef Mezouar, and Philippe Martinet Research Paper in this link

Unified Projection Model Based on Double Projection through a Unitary Sphere

Geyer et al. [Geyer 2000] proposed a unified projection model that revolutionizes the representation of panoramic images. This model is founded on the concept of double projection through a unitary sphere, providing a comprehensive framework for mapping 2D images onto spherical surfaces.

Central imaging systems can be modeled using two consecutive projections: spherical projection succeeded by a perspective one. This geometric formulation called unified model has been proposed by Geyer and Daniilidis in [**?**] and has been intensively used by the vision and robotics community (structure from motion, calibration, visual servoing, etc).

Consider the virtual unitary sphere centered in the origin of the mirror frame F_m as shown in Fig. 1 and the perspective camera centered in the origin of the camera frame F_c. Without loss of generality, a simple translation of $-\xi$, along the Z axis of the camera frame, can be used to normalize the image points. The projection model can be described as follows:

1. A 3D point X_s on the unit sphere is projected onto the normalized image plane $Z = 1 - \xi$ into a point of homogeneous coordinates:

$$x = f(X_s) = \begin{bmatrix} X/Z + \xi\rho \\ Y/Z + \xi\rho \\ 1 \end{bmatrix} \tag{1}$$

   where ρ is the mirror parameter, X, Y, Z are the 3D coordinates of the point X_s, and $Z \neq 0$.

2. The 2D projective point x is mapped into the pixel image point with homogeneous coordinates x_i using the collineation matrix K:

$$x_i = Kx \tag{2}$$

   where K contains the conventional camera intrinsic parameters coupled with mirror intrinsic parameters, and can be written as:

$$K = \begin{bmatrix} f_u & \alpha_{uv} & u_0 \\ 0 & f_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3}$$

3. The inverse projection from the image plane onto the unit sphere can be obtained by inverting the second and last steps. The point onto the unit sphere is then obtained by inverting the nonlinear projection:

$$X_s = f^{-1}(x) = \eta \begin{bmatrix} x \\ y \\ 1 - \xi/\eta \end{bmatrix} \tag{4}$$

   where $\eta = \xi + \sqrt{1 + (1 - \xi^2)(x^2 + y^2)} / \sqrt{x^2 + y^2 + 1}$.

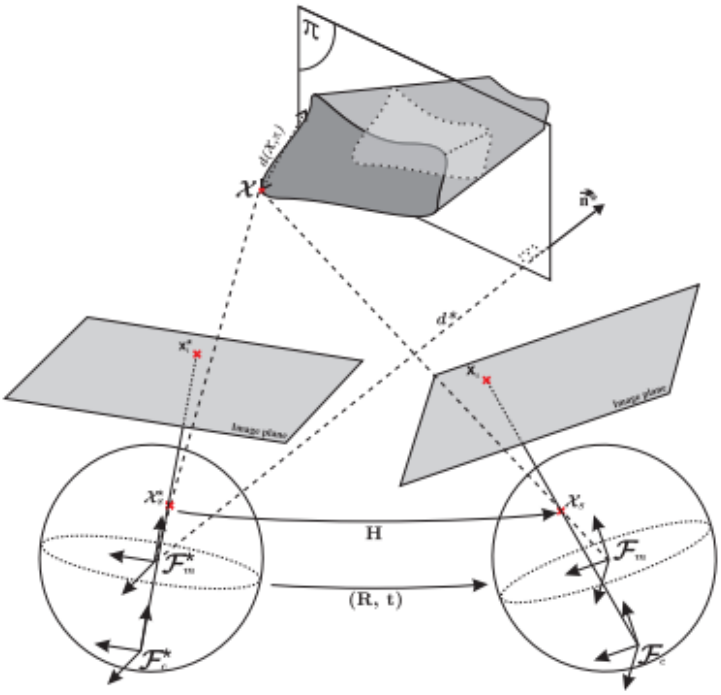


Figure 1: Unified central projection and two views geometry.

To obtain Euclidean reconstruction from two views, several methods have been proposed based on the estimation of essential or homography matrices. For control purposes, it is preferable to use methods based on the homography matrix, as they are less susceptible to degenerate configurations. The homographic relationship between two central views of points can be computed as follows:

Consider two positions F_m and $F_{m'}$ of the central camera, related by the rotation matrix R and the translation vector t. Let (π) be a 3D reference plane given in $F_{m'}$ by the vector $\pi' = [n' - d']$, where n' is its unit normal in $F_{m'}$ and d' is the distance from (π) to the origin of $F_{m'}$. Let X be a 3D point with coordinates $X = [X, Y, Z]$ with respect to F_m and coordinates $X' = [X', Y', Z']$ with respect to $F_{m'}$. The homography matrix H related to the plane (π) can be estimated up to a scale factor by solving the linear equation $X_s \times HX'_s = 0$ using at least four couples of coordinates (X_s, X'_s), corresponding to the spherical projection of world points X belonging to (π). The homography matrix H can be written as a function of the camera displacement and of the plane coordinates with respect to $F_{m'}$, and has the same form as in the conventional perspective case. The relationship between the coordinates of X with respect to F_m and $F_{m'}$ can be written as a function of their spherical coordinates:

$$\rho X_s = \rho' R X'_s + t \tag{5}$$

By multiplying and dividing the translation vector by the distance d' and according to the homography matrix H, the expression can be rewritten as:

$$\rho X_s = \rho' H X'_s + \alpha t \tag{6}$$

where $H = R + td'^{-1}n'$ is the Euclidean homography matrix, and $\alpha = -d(X, \pi)/d'$ is a scaling factor. The homography matrix H can be used to obtain the Euclidean reconstruction of the 3D points.

Mapping from projective space to the sphere to an image plane: A point in projective space is first projected to an antipodal point pair on the sphere. An axis of the sphere is chosen, as well as a point on this axis, denoted as the projection center. The projection is defined as a mapping from the sphere to the image plane, where the projection center is on a sphere diameter and the plane is perpendicular to it .

The projection of the great circle to the image plane is illustrated in Figure 1. The equator is projected to a circle of radius $\frac{l+m}{l}$, which represents the horizon of the fronto-parallel, as the equator is the projection of the line at infinity in the plane $z = 0$.

For the necessary equations, you can take look into the research paper [2].

2 Into the practical part

2.1 Exercise 1

1. The `ImToSphere` function transforms a 2D image into spherical coordinates using panoramic camera calibration and two projection methods. It takes inputs: source image (`I`), calibration matrix (`K`), mirror parameter (`csi`), spherical image dimension (`imsph_dim`), and projection method. The output includes the spherical image (`out`) and angle vectors (`phi_vec` and `theta_vec`).

   The function performs either omnidirectional or Daniilidis projection based on the specified method:

   - **Omnidirectional Projection (Method 1):**
     - Converts spherical to Cartesian coordinates.
     - Applies omnidirectional projection using mirror parameter.
     - Projects onto the 2D image plane using calibration matrix.
   - **Daniilidis Projection (Method 2):**
     - Calculates 2D coordinates using Daniilidis method.
     - Forms a 3xN matrix with homogeneous coordinates.
     - Projects onto the 2D image plane using calibration matrix.

   The function then performs interpolation to create the spherical image. NaN values resulting from interpolation are set to 0.

   Usage: `[out,` $\phi\_vec, \theta\_vec] = ImToSphere(I, K, csi, imsph\_dim, method)$

2. Warpping an omnidirectional image onto a spherical image involves projecting the 2D image onto the surface of a unitary sphere. This process aims to achieve a panoramic representation by mapping pixels from a planar image onto the curved surface of a sphere.

   We chose the image shown in the Fig.2 as a 2D omnidirectional image, then we have had as an output the image shown in Fig.3 as a Spherical output of our function.

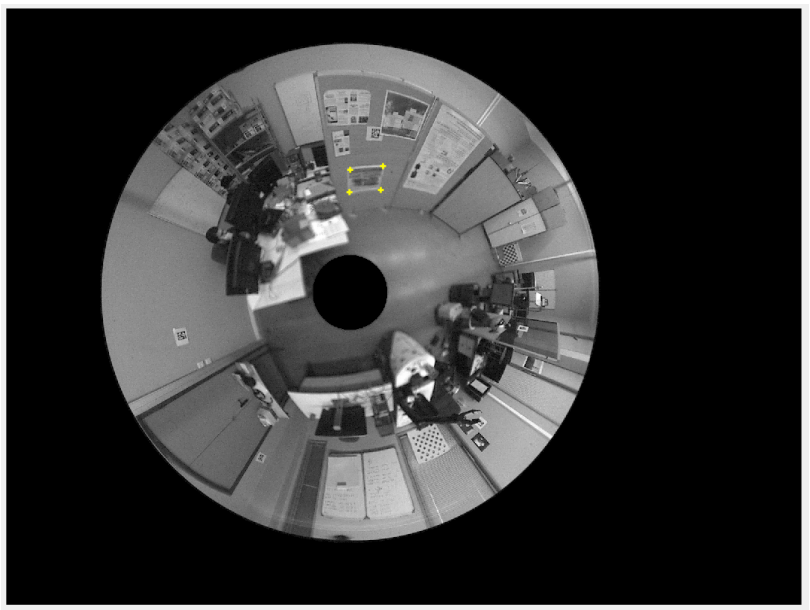
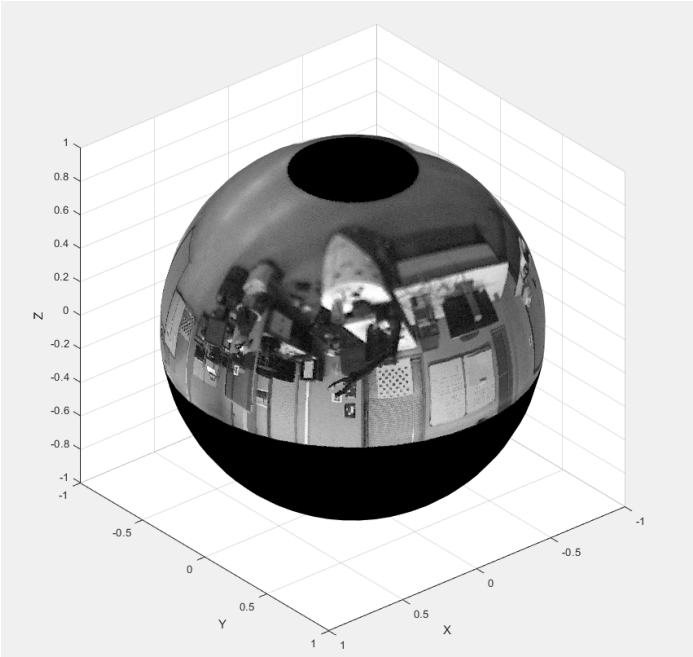


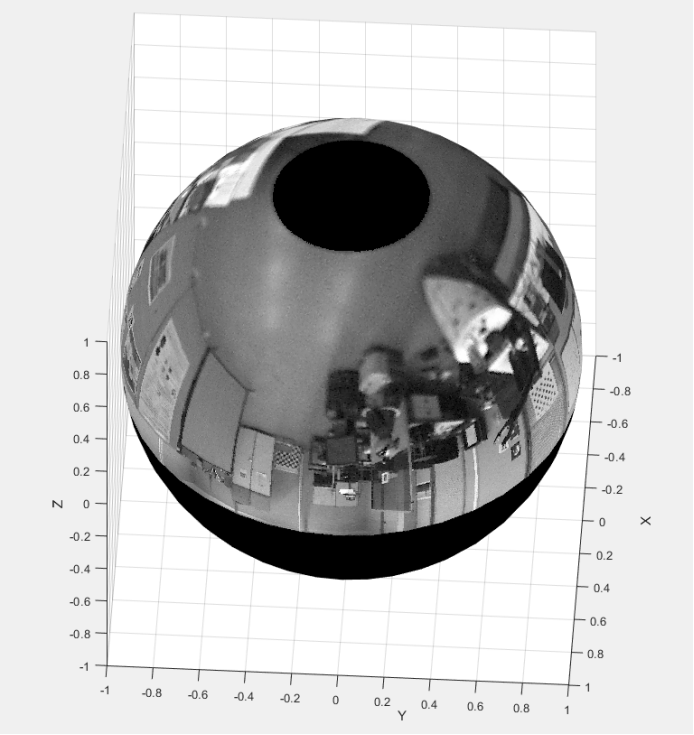
Figure 2: The 2D omni image

Discussion on Spherical Image Results:

The avoidance of the north and south poles in the spherical images generated by the `ImToSphere` function can be explained by the nature of panoramic projections. Panoramic images often employ projections that distort polar regions due to the convergence of meridians. This distortion becomes more pronounced towards the poles, leading to less accurate representation.



(a) Spherical Image output view 1



(b) Spherical Image output view 2

Figure 3: Spherical Image output

The avoidance of the poles is a common strategy in panoramic imaging to mitigate distortion and maintain a more visually appealing and useful representation. The mathematical transformations involved in the `ImToSphere` function, including spherical coordinate conversions and projections, contribute to this avoidance.

Adjusting projection parameters or employing specialized projection techniques can sometimes mitigate pole avoidance, but it often involves trade-offs in other aspects of the image representation.

3. To generate an omnidirectional image from a 3D spherical image, a reverse process is applied, involving the conversion of spherical coordinates to Cartesian coordinates and then projecting onto a 2D plane.

   Let (ϕ, θ) be the spherical coordinates representing points on the unitary sphere, where ϕ is the azimuthal angle and θ is the polar angle. The conversion from spherical to Cartesian coordinates is given by:

$$X = \sin(\theta)\cos(\phi)$$
$$Y = \sin(\theta)\sin(\phi)$$
$$Z = \cos(\theta)$$

   The Cartesian coordinates (X, Y, Z) are then projected onto a 2D plane using a suitable projection method, such as omnidirectional projection. This projection depends on the specific characteristics of the omnidirectional camera used and may involve distortion correction.

2.2 Exercise 2

Discussion on Homography Matrix and Perspective Transformation:

In computer vision, a homography matrix is a transformation that relates points in one image to corresponding points in another image. It represents a perspective transformation that captures the geometric relationship between two images taken from different viewpoints or under different camera projections.

The homography matrix is particularly useful for tasks such as image stitching, object recognition, and augmented reality. It allows the mapping of points between images even when there are variations in scale, rotation, and perspective.

The practical implementation in the `main_prg_Homography_Perspective.m` MATLAB file involves the application of homography to warp 2D points from one image to another. This process enables the alignment of features or objects in different images, facilitating further analysis or synthesis.

The homography matrix is derived based on corresponding points manually or through feature matching algorithms. Once obtained, it can be used to transform points or entire images, providing a powerful tool for tasks involving multiple views or perspectives.

Calculating a Homography Matrix:

The homography matrix (H) is calculated based on corresponding points in two images. Given a set of at least four pairs of corresponding points (x_i, y_i) in one image and (x'_i, y'_i) in another image, the homography matrix can be computed using a linear system of equations.

The homography matrix H satisfies the equation:

$$s \begin{bmatrix} x'_i \\ y'_i \\ 1 \end{bmatrix} = H \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

This equation can be expressed as a set of linear equations for each corresponding point pair. Collecting these equations for all pairs, we form the matrix equation:

$$\begin{bmatrix} x'_1 & y'_1 & 1 & 0 & 0 & 0 & -x_1x'_1 & -x_1y'_1 \\ 0 & 0 & 0 & x'_1 & y'_1 & 1 & -y_1x'_1 & -y_1y'_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x'_n & y'_n & 1 & 0 & 0 & 0 & -x_nx'_n & -x_ny'_n \\ 0 & 0 & 0 & x'_n & y'_n & 1 & -y_nx'_n & -y_ny'_n \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix} = 0$$

This system of linear equations is solved using methods such as singular value decomposition (SVD). The solution is a vector containing the elements of the homography matrix. The vector is then reshaped to obtain the 3x3 homography matrix.

Note

In MATLAB, the function `fitgeotrans` or `estimateGeometricTransform` can be used for homography estimation, providing a convenient way to calculate the homography matrix.

Explanation of the `homography_plan` Function:

The `homography_plan` function computes a homography matrix (`h`) based on corresponding points in two images. The mathematical operations performed in the function can be explained using the LaTeX algorithm environment:

Algorithm 1 `homography_plan` Function

function HOMOGRAPHY_PLAN(p, p_d)
  Initialize matrix M and vectors u, s, v
  for $j = 1$ to m **do**
    Fill matrix M with values based on corresponding points p and p_d
  end for
  Perform Singular Value Decomposition (SVD) on M: $[u, s, v] = \text{svd}(M)$
  Extract the last column of v to obtain the homography vector
  Reshape the homography vector to obtain the 3×3 homography matrix h
  return h
end function

In each iteration of the loop, the function populates the matrix M using the provided corresponding points. The SVD is then applied to M, and the last column of v contains the elements of the homography vector. This vector is reshaped to form the 3×3 homography matrix h, which is returned by the function.

After running **main_prg_Homography_Perspective.m** in Matlab, we have got the following resuilts in Fig.4.

The numerical results for the homography matrices (H_A and H_B) obtained using the `homography_plan` function are as follows:

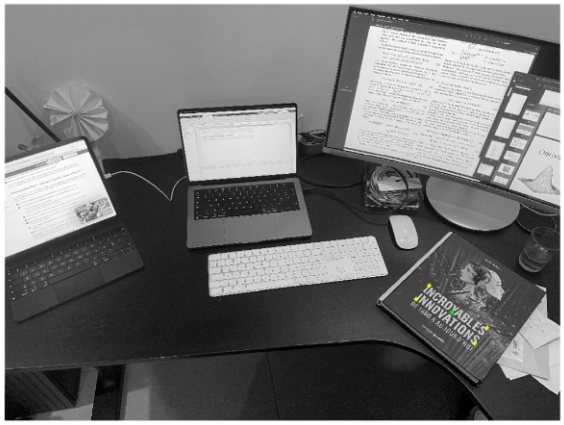
For Plane A:

$$H_A = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} = \begin{bmatrix} -0.0021 & 0.00 & 0.03 \\ -0.001 & 0.001 & 0.9994 \\ 0.000 & 0.000 & -0.0023 \end{bmatrix}$$

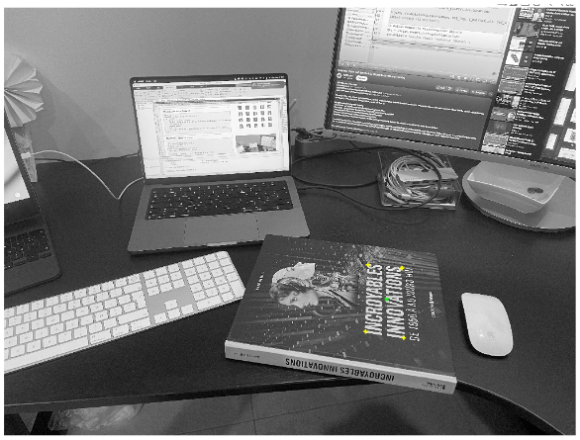
For Plane B:

$$H_B = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} = \begin{bmatrix} 0.0027 & -0.0011 & 0.0333 \\ -0.001 & 0.0016 & 0.9994 \\ 0.000 & 0.000 & 0.0032 \end{bmatrix}$$

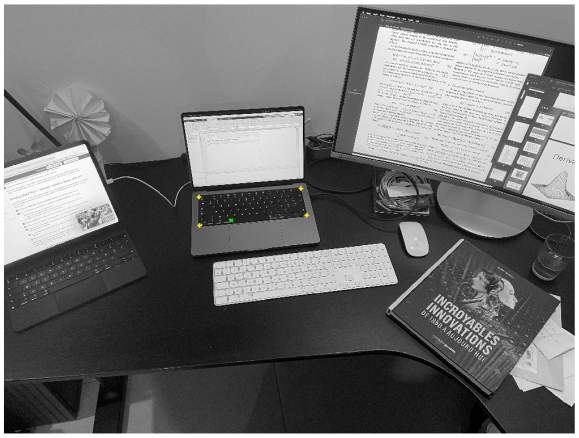
These 3x3 matrices represent the homography transformations between the corresponding points on each plane. The elements h_{ij} capture the linear relationship between the coordinates of the points in the two planes.



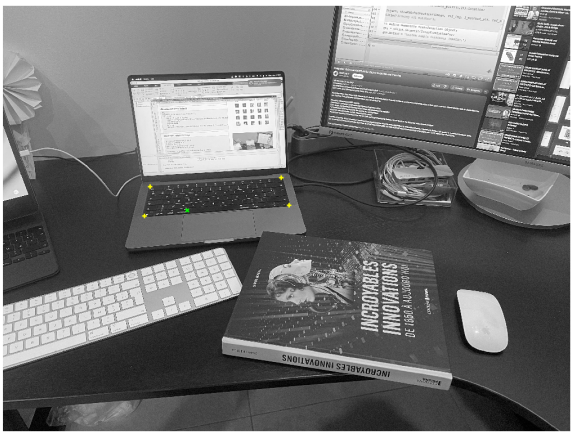
(a) 2D Image with four training features (in yellow) and the feature to be detected (in green) for Plane A



(b) **main_prg_Homography_Perspective.m** output put



(c) 2D Image with four training features (in yellow) and the feature to be detected (in green) for Plane B



(d) **main_prg_Homography_Perspective.m** output put

Figure 4: Testing **main_prg_Homography_Perspective.m** for two diffrent plane; Plane A (a) and (b) ; Plane B (c) and (d)

3 Exercise 3

In this exercise, the objective is to estimate the homography relationship between two omnidirectional images using a function initially designed for conventional cameras. The process involves converting 2D image points into metric coordinates, back-projecting them into the spherical space, and establishing the geometric relationship between two omnidirectional images.

1. **Conversion to Metric Coordinates:** Initially, 2D image points are converted from pixel coordinates to metric ones. This step ensures that the points are represented in a consistent metric space, providing a foundation for subsequent calculations.

2. **Back-Projection to Spherical Space:** The next step involves back-projecting the 2D normalized points into the spherical space. This is achieved by inversing the unified projection model, allowing the conversion of 2D metric coordinates and mirror parameters into spherical coordinates. The result is the representation of feature points in the omnidirectional image within the spherical space.

Before establishing the homography relationship (or epipolar constraint) between two images, it is crucial to calibrate the omnidirectional camera and generate spherical images from corresponding 2D images. This calibration ensures the accuracy of subsequent transformations.

Later in the process, when choosing a feature point in the 2D omnidirectional image, the function should ideally yield the same feature point in the corresponding spherical image. This verification step helps validate the accuracy of the back-projection process and ensures that the geometric relationship between the omnidirectional images is appropriately captured.

Estimating Homography Relationship for Omnidirectional Images

The process of estimating the homography relationship between two omnidirectional images involves mathematical transformations to convert 2D image points into the spherical space. Here are the key mathematical steps:

1. **Conversion to Metric Coordinates:** Given a 2D image point (u, v), it needs to be converted from pixel coordinates to metric coordinates. This involves applying the inverse of the camera calibration matrix to obtain (x, y, z) in metric space.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = K^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

2. **Back-Projection to Spherical Space:** The next step is to back-project the 2D normalized point into the spherical space. This is achieved by inverting the unified projection model. The unified projection model, as proposed by Geyer et al., involves projecting points through a unitary sphere.

   The inverse transformation from 2D metric coordinates (x, y, z) and mirror parameter csi to spherical coordinates (ϕ, θ) can be expressed using appropriate equations based on the specific model used.

3. **Homography Relationship:** Once the spherical coordinates are obtained for feature points in both omnidirectional images, the homography relationship (or epipolar constraint) can be established. This involves finding the transformation matrix H that relates corresponding points in the spherical space of the two images.

The accuracy of this process relies on proper camera calibration, correct application of the inverse unified projection model, and successful validation by checking if the chosen feature point in the 2D omnidirectional image corresponds accurately to the same point in the spherical image.

Displaying Points on Spherical Images using `ginput` and `yashow`:

The given code snippet defines the `yashow` MATLAB function, which is designed for displaying various types of data, including matrices, vectors, volumes, and spherical data. This function enables visualization in different ways based on the input data type.

The `yashow` function is versatile and can adapt to the type of data provided. For the case of spherical data (`type = 'spheric'`), it calls the `yashow_spheric` function to handle the specific visualization. The code includes provisions for displaying volumes, matrices, vectors, and time sequences as well.

Now, let's define the `ginput` function:

The `ginput` function is a MATLAB function that allows interactive selection of points from a figure using the mouse. When called, it waits for the user to click on the figure, and the selected point's coordinates are returned.

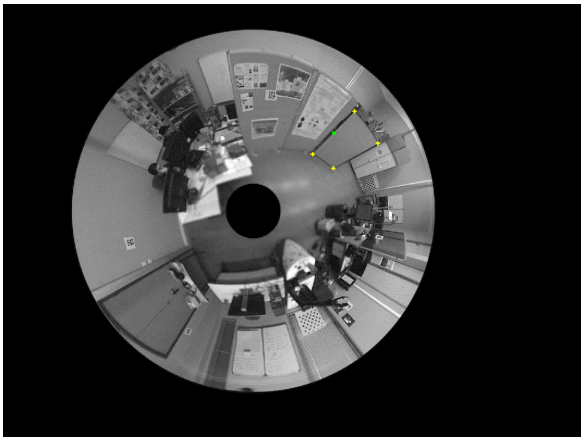
`yashow` Function:

Algorithm 2 `yashow` Function

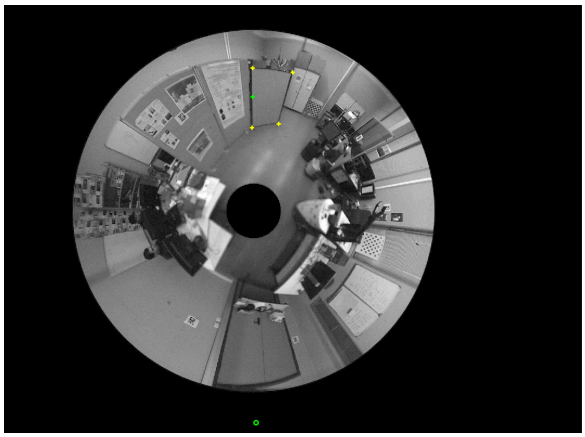
```
function YASHOW(yastruct, ...)
    if is numeric or logical data then
        Handle as volume, matrix, vector, or time sequence based on dimensions
    else if is a structure with type field then yastruct.type 'volume'
        Handle as volume using yashow_volume 'matrix'
        Handle as matrix using yashow_matrix 'spheric'
        Handle as spherical data using yashow_spheric 'sphvf'
        Handle as spherical vector field using yashow_sphvf 'vector'
        Handle as a vector, plot with plot 'timeseq'
        Handle as a time sequence using yashow_timeseq
        Error: Unrecognized input type
    else
        Error: Unrecognized input
    end if
end function
```

3.0.1 Important Note

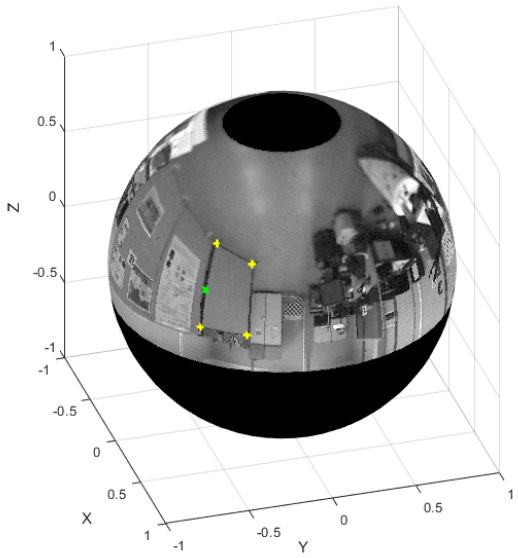
Our main function for generating the same 2D features in the second 3D spherical image, while running, is generating **two diffrent** solutions, and the difference between these two solutions is π **= 180°**, one of them is the same feature in the spherical image, and one of them is supposed to be wrong. So, in the following figures you will see both of the solutionswhen we chage the image and try to find the same feature.



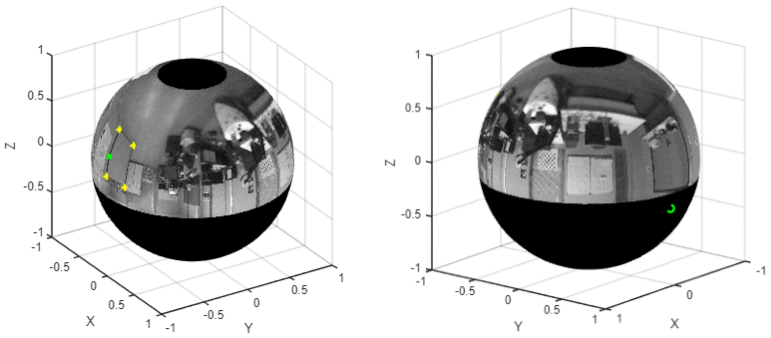
(a) 2D Image (1) with four training features (in yellow) and the feature to be detected (in green)



(b) The same detected feature in 2D omnidirectional Image(2)



(c) Output of Image(1) with the detected feature



(d) Output of Image(2) with the detected feature, right solution (x), wrong solution (o)

Figure 5: Testing the main function on detecting the same feature from 2D Omnidirectional image to spherical Image

4 Exercise 4

Generating Perspective Image from Spherical Image:

To create a perspective image from a spherical image, the following detailed solution involves selecting a 2D point in the omnidirectional image, converting it to metric and spherical coordinates, and then performing a perspective projection.

1. Selecting a 2D Point:

Use the `ginput` function to interactively select a 2D point (u, v) from the omnidirectional image. This point will act as the center for the perspective view.

2. Conversion to Metric Coordinates:

Convert the 2D point to metric coordinates (x, y, z) using the inverse of the camera calibration matrix:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = K^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

3. Back-Projection to Spherical Space:

Apply the inverse of the unified projection model to obtain spherical coordinates (ϕ, θ) from metric coordinates and the mirror parameter (csi):

$$(\phi, \theta) = \text{InverseProjection}(x, y, z, csi)$$

4. Define Perspective Parameters:

Define parameters for the perspective transformation, including the field of view (FOV) and the aspect ratio (AR).

5. Perspective Projection:

Using the selected point in spherical coordinates and the defined perspective parameters, perform the perspective projection to obtain perspective coordinates (u', v'):

$$\begin{bmatrix} u' \\ v' \end{bmatrix} = \text{PerspectiveProjection}(\phi, \theta, \text{FOV}, \text{AR})$$

6. Interpolation:

Use interpolation techniques (e.g., bilinear interpolation) to obtain pixel values for the perspective image from the spherical image around the computed (u', v') coordinates.

7. Display Perspective Image:

Use the `imshow` function to display the generated perspective image.

Equations:

Inverse Projection:

$$(\phi, \theta) = \text{InverseProjection}(x, y, z, csi)$$

Perspective Projection:

$$\begin{bmatrix} u' \\ v' \end{bmatrix} = \text{PerspectiveProjection}(\phi, \theta, \text{FOV}, \text{AR})$$

Note

This part may be wrong since it is just a proposition for the exerrcise 4, the practical part of this exercise is not done, and the functions **InverseProjection** and **PerspectiveProjection** are just theoritically explained, no code is included for such functions.

5 Lab Summary

In this lab on non-conventional cameras and image transformations, we explored various concepts and techniques, enhancing our understanding of computer vision. Here's a concise summary of the key points observed throughout the lab:

Camera Projection and Calibration:

We delved into the theoretical aspects of camera projection, defining the projection matrix P and intrinsic parameters. The Single Viewpoint Constraint was emphasized, providing a foundation for camera calibration.

Unified Projection Model:

Geyer et al.'s unified projection model introduced a novel approach based on twice projecting through a unitary sphere. This model forms the basis for moving from 2D images to spherical images, involving intricate mathematical transformations.

Conversion to Spherical Coordinates:

The provided MATLAB function `ImToSphere` demonstrated the conversion of 2D Cartesian coordinates into spherical coordinates. It utilized known techniques such as omnidirectional projection and Daniilidis method, showcasing the practical implementation of theoretical concepts.

Homography and Feature Detection:

The lab extended to feature detection in both 2D and spherical images, employing homography matrices for accurate correspondence. The use of grayscale scaling enhanced feature matching, and the application of homography allowed us to relate points between different perspectives.

Perspective Image Generation:

A proposed solution detailed the generation of a perspective image from a spherical image. By selecting a 2D point, converting to metric and spherical coordinates, and applying perspective projection, we achieved a novel way to create perspective views.

Interactive Visualization:

The `ginput` function facilitated interactive point selection, enhancing user engagement. Additionally, the `yashow` function provided a versatile tool for displaying various types of data, contributing to effective visualization.

Analytical Development:

Throughout the lab, we emphasized analytical development, connecting theoretical concepts with practical implementations. The utilization of MATLAB functions showcased our ability to translate mathematical understanding into executable code.

This lab served as a comprehensive exploration of non-conventional cameras, offering valuable insights into image transformations and computer vision techniques. Our analytical approach and hands-on experience furthered our grasp of the subject matter, paving the way for deeper exploration in future studies.

6 General Conclusion

The non-conventional cameras lab provided a rich exploration into the intricate world of computer vision, covering fundamental theoretical concepts and practical implementations. Through a series of exercises, we gained insights into camera projection, calibration, and the unified projection model proposed by Geyer et al.

The conversion from 2D Cartesian coordinates to spherical coordinates, as demonstrated in the `ImToSphere` function, highlighted the complexities involved in mapping images onto a spherical surface. We utilized established techniques such as omnidirectional projection and homography matrices for feature detection, enhancing our understanding of image transformation processes.

The proposed solution for generating perspective images from spherical images introduced a novel approach, involving interactive point selection, metric conversions, and perspective projection. This not only broadened our skill set but also emphasized the practical applications of theoretical knowledge.

In conclusion, this lab not only deepened our understanding of non-conventional cameras and image transformations but also honed our analytical and practical skills in computer vision. The integration of theory and implementation in MATLAB showcased the power of bridging mathematical concepts with real-world applications. This foundational knowledge lays the groundwork for continued exploration and advancements in the dynamic field of computer vision.

7 Refrences

1. **A Unifying Theory for Central Panoramic Systems and Practical Implications** Christopher Geyer and Kostas Daniilidis University of Pennsylvania, GRASP Laboratory, Pennsylvania, PA 19104 Research Paper in this link

2. **Points-Based Visual Servoing with Central Cameras** Hicham Hadj-Abdelkader, Youcef Mezouar, and Philippe Martinet Research Paper in this link

3. **Self-calibration for consumer omnidirectional multi-camera mounted on a helmet** Thanh-Tin NguyenResearch Paper in this link

Annex

The following code given by the professor Hicham HADJ-ABDELKADER :

```
clear all;
close all;

% Camera paramaters
K= [425.19303  0       692.86729;
0      424.86463  572.11922;
0      0       1];
csi=0.98754;
Bw=2*512;


%% Image 1
% Load and normalize the image 1
tmp=imread('images/Im_R0_T0.pgm') ;
img=double(tmp(:,:,1));
I1=img/max(img(:));
figure(1);
imshow(I1); hold on;

% Extract features from image 1 (4 point)
figure(1);
uv1 = zeros(3,4);
for i=1:4
    tmp = ginput(1);
    uv1(:,i) = [tmp'; 1];
    plot(uv1(1,i),uv1(2,i),'+y','LineWidth',2);
end;
% warp the image point in pixel to the spherical point
% First, you have to warp the pixelic coordinates to the normalized one
% using the calibration matrix
% The, you can use the function inv_omniproj provided in Tools folder
% ... code goes here
% ................................
% ................................

% Generate the spherical image
[Is1,phi_vec,theta_vec]=ImToSphere(I1,K,csi,Bw,1);

% draw Spherical point onto the spherical image
figure(3);
yashow(Is1,'Spheric'); colormap gray; hold on;
plot3(Is1(1,:),Is1(2,:),Is1(3,:),'+y', 'LineWidth',2);



%% image 2
% Load and normalize the image 2
tmp=imread('images/Im_R45_T0.pgm') ;
img=double(tmp(:,:,1));
I2=img/max(img(:));
figure(2);
imshow(I2); hold on;
```

```
% Extract features from image 1
figure(2);
uv2 = zeros(3,4);
for i=1:4
    tmp = ginput(1);
    uv2(:,i) = [tmp'; 1];
    plot(uv2(1,i),uv2(2,i),'+y','LineWidth',2);
end;
% Do the same job as before (as done in the image 1)

% Generate the spherical image
[Is2,phi_vec,theta_vec]=ImToSphere(I2,K,csi,Bw,1);

% draw Spherical point onto the spherical image
figure(4);
yashow(Is2,'Spheric'); colormap gray; hold on;
plot3(Is2(1,:),Is2(2,:),Is2(3,:),'+y', 'LineWidth',2);


%% Homography
% compute the homography matrix using the spherical points rather than 2D
% image point used for perspective images
% .... code goes here
% ..............................

% test Homography
figure(1);
p1 = [ginput(1) 1]';
plot(p1(1),p1(2),'xg','LineWidth',2);

% generate the corresponding spherical point s1
% .... the code goes here
% ..............................
% ..............................
% plot the spherical point onto the spherical image
figure(3);
plot3(Is1(1,:),Is1(2,:),Is1(3,:),'xg', 'LineWidth',2);

% warp s1 to s2 through the homography matrix
% ... code goes here
% ................

% Note that when s1 is warped onto the second spherical image, two
% antipodal points are obtaied. Indeed, s2 and -s2 satisfied both the homographic
% relationship
% first spherical point
Is2 = Is2/norm(Is2);
x2 = omniproj(Is2, csi);
p2 = K*x2;
figure(2);
plot(p2(1),p2(2),'+g','LineWidth',2);
% antipodal point
x2 = omniproj(-Is2, csi);
p2 = K*x2;
figure(2);
plot(p2(1),p2(2),'og','LineWidth',2);

figure(4);
plot3(Is2(1,:),Is2(2,:),Is2(3,:),'xg', 'LineWidth',2);
plot3(-Is2(1,:),-Is2(2,:),-Is2(3,:),'og', 'LineWidth',2);
```

The above code has been changed to:

```
clear all;
close all;

% Camera paramaters
K= [425.19303  0       692.86729;
0      424.86463  572.11922;
0      0       1];
csi=0.98754;
Bw=2*512;


%% Image 1
% Load and normalize the image 1
tmp=imread('images/Im_R0_T0.pgm') ;
img=double(tmp(:,:,1));
I1=img/max(img(:));
figure(1);
imshow(I1); hold on;

% Extract features from image 1 (4 point)
figure(1);
uv1 = zeros(3,4);
for i=1:4
    tmp = ginput(1);
    uv1(:,i) = [tmp'; 1];
    plot(uv1(1,i),uv1(2,i),'+y','LineWidth',2);
end;
% warp the image point in pixel to the spherical point
% First, you have to warp the pixelic coordinates to the normalized one
% using the calibration matrix
% Then, you can use the function inv_omniproj provided in Tools folder
% ... code goes here
xy1 = K\uv1; % equivalent to inv(K)*uv1
S1 = inv_omniproj(xy1, csi);


% Generate the spherical image
[Is1,phi_vec,theta_vec]=ImToSphere(I1,K,csi,Bw,1);

% draw Spherical point onto the spherical image
figure(3);
yashow(Is1,'Spheric'); colormap gray; hold on;
plot3(S1(1,:),S1(2,:),S1(3,:),'+y', 'LineWidth',2);



%% image 2
% Load and normalize the image 2
tmp=imread('images/Im_R45_T0.pgm') ;
img=double(tmp(:,:,1));
I2=img/max(img(:));
figure(2);
imshow(I2); hold on;
```

```
figure(2);
uv2 = zeros(3,4);
for i=1:4
    tmp = ginput(1);
    uv2(:,i) = [tmp'; 1];
    plot(uv2(1,i),uv2(2,i),'+y','LineWidth',2);
end
% Do the same job as before (as done in the image 1)
xy2 = K\uv2; % equivalent to inv(K)*uv1
S2 = inv_omniproj(xy2, csi);
% Generate the spherical image
[Is2,phi_vec,theta_vec]=ImToSphere(I2,K,csi,Bw,1);
% draw Spherical point onto the spherical image
figure(4);
yashow(Is2,'Spheric'); colormap gray; hold on;
plot3(S2(1,:),S2(2,:),S2(3,:),'+y', 'LineWidth',2);

%% Homography
% compute the homography matrix using the spherical points rather than 2D
% image point used for perspective images
H = homography_plan(S2,S1);

% test Homography
figure(1);
p1 = [ginput(1) 1]';
plot(p1(1),p1(2),'xg','LineWidth',2);

% generate the corresponding spherical point s1
% .... the code goes here
x1 = K\p1; % equivalent to inv(K)*uv1
s1 = inv_omniproj(x1, csi);

% plot the spherical point onto the spherical image
figure(3);
plot3(s1(1,:),s1(2,:),s1(3,:),'xg', 'LineWidth',2);

% warp s1 to s2 through the homography matrix
% ... code goes here
s2 = H*s1;

% Note that when s1 is warped onto the second spherical image, two
% antipodal points are obtaied. Indeed, s2 and -s2 satisfied both the homographic relationship
% first spherical point
s2 = s2/norm(s2);
x2 = omniproj(s2, csi);
p2 = K*x2;
p2 = p2/p2(3);
figure(2);
plot(p2(1),p2(2),'+g','LineWidth',2);
% % antipodal point
x2 = omniproj(-s2, csi);
p2 = K*x2;
p2 = p2/p2(3);
figure(2);
plot(p2(1),p2(2),'og','LineWidth', 2);

figure(4);
plot3(s2(1,:),s2(2,:),s2(3,:),'xg', 'LineWidth',2);
plot3(-s2(1,:),-s2(2,:),-s2(3,:),'og', 'LineWidth',2);
```